数据科学基础
(With Python)

模型评价

模型评价(1)

- 给定一个机器学习问题(如分类)
- 打算用模型M解决问题
  - 是个函数空间,如y=wx+b之类
- 利用训练集D_1,训练出M_1
  - 是个具体函数,如y=3x+2
- 对M和M_1的评价方法是不同的.

模型评价(2)

- 用M_1预测测试样本x(对应类别标签y),返回预测值y_1
  - 误差$y-y_1$可用来评价M_1,但不足以评价M
- 再利用训练集D_2,训练出M_2
  - M_2对x预测y_2, $y-y_2$可用来评价M_2,但不足以评价M

模型评价(3)

- 利用不同训练集训练n次,得到n个模型 $M_1, ..., M_n$ $\bar{y}$
  - 对同一测试样本x分别预测$y_1, ..., y_n$
  - 当n足够大,即可利用$y_1, ..., y_n$评价M

模型评价(4)

- 先看n个预测值y_i的均值 $\bar{y}$ 是否等于y
  - 是:则M学习能力足够,预测的期望值是对的
  - 否:则M学习能力有缺陷,无法通过反复训练解决
    - 例如,M只有一个函数f,总对x的类别预测为y+1
- 因学习能力不够带来的预测误差称为偏差

模型评价(5)

- 那么,如果M和M′经过足够多次训练都能使预测值均值等于y,是否两者就一样好?
  - No!因为实际中不大会有很多个训练集来尝试,往往只有一个或几个训练集.
- 需要看n个预测值的方差
  - 方差大,说明预测值虽然围绕 $\bar{y}$,但散布广
  - 方差小,说明预测值集中于 $\bar{y}$ 周围

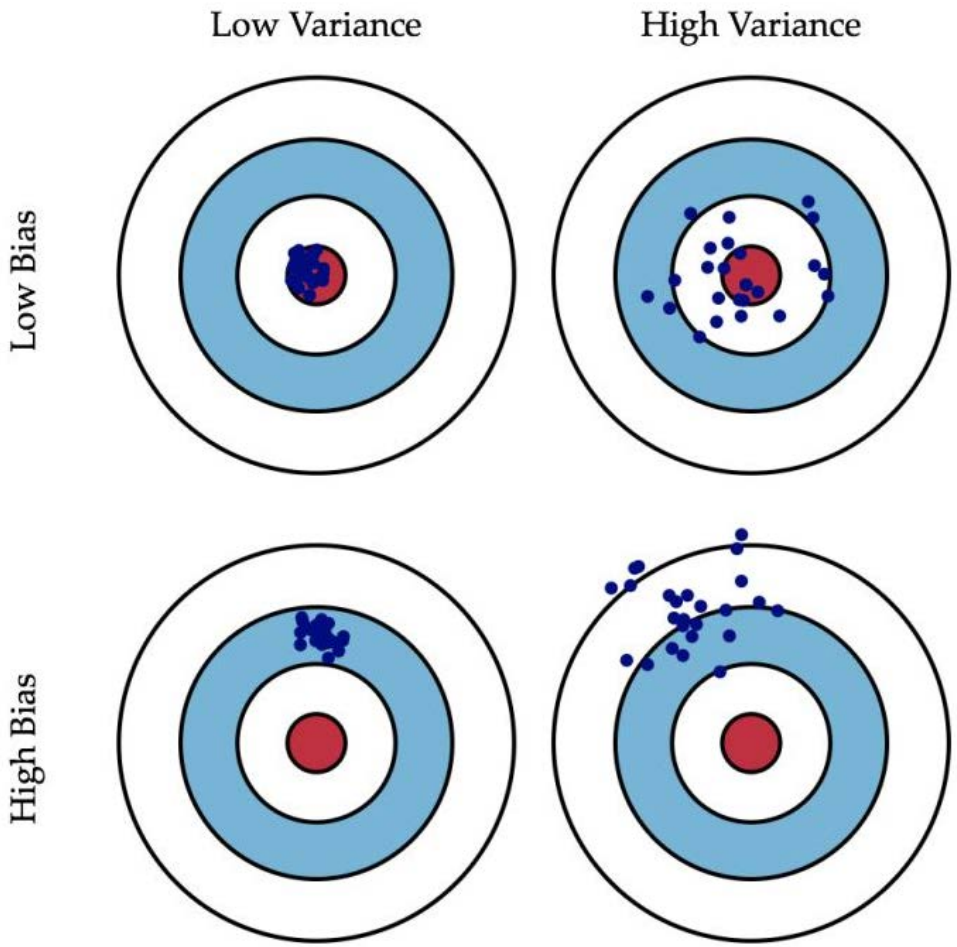
偏差

- 偏差:模型的期望预测值与实际值之差.
  - 基于正确率来度量模型.
  - 简单说就是预测值与实际值之差.
- 如果模型一般预测很准确,则说它是低偏差模型;如果经常出错,则说它是高偏差模型.
- 偏差越大,越偏离真实值.

方差

- 方差描述的是预测值的变化或离散程度,即预测值与期望值的距离.
- 方差越大,数据的分布越分散.
- 考虑从同一个总体反复随机抽样,方差衡量模型每次拟合的变化有多大.
- 如果模型每次变化不大,则模型是低方差模型;如果对不同样本变化很大,则说是高方差模型

理想:低偏差低方差



两种极端情形

- 欠拟合:模型拟合数据时做得不细.
  - 高偏差低方差的模型易于欠拟合.
  - 案例中,线性回归对数据集欠拟合.
- 过拟合:模型拟合数据时过于精细
  - 低偏差高方差的模型易于过拟合
  - 案例中,四次多项式回归对数据集过拟合.

解决方法

- 如果模型显示高偏差或欠拟合,则可:
  - 使用更多特征:纳入新特征以改进预测能力
  - 尝试更复杂的模型:增加模型复杂度有助于改进偏差.
    - 当然,过于复杂的模型也不好.
- 如果模型显示高方差或过拟合,则可:
  - 使用较少特征:可以降低方差防止过拟合
  - 拟合更多训练样本,在交叉验证中使用更多训练样本可减小过拟合的作用,改进高方差.

偏差/方差与误差函数

- 误差函数度量模型的不正确性,可视为偏差,方差和不可归约误差的函数.

$$E((y-\hat{y})^2) = Bias(\hat{y})^2 + Var(\hat{y}) + \varepsilon$$

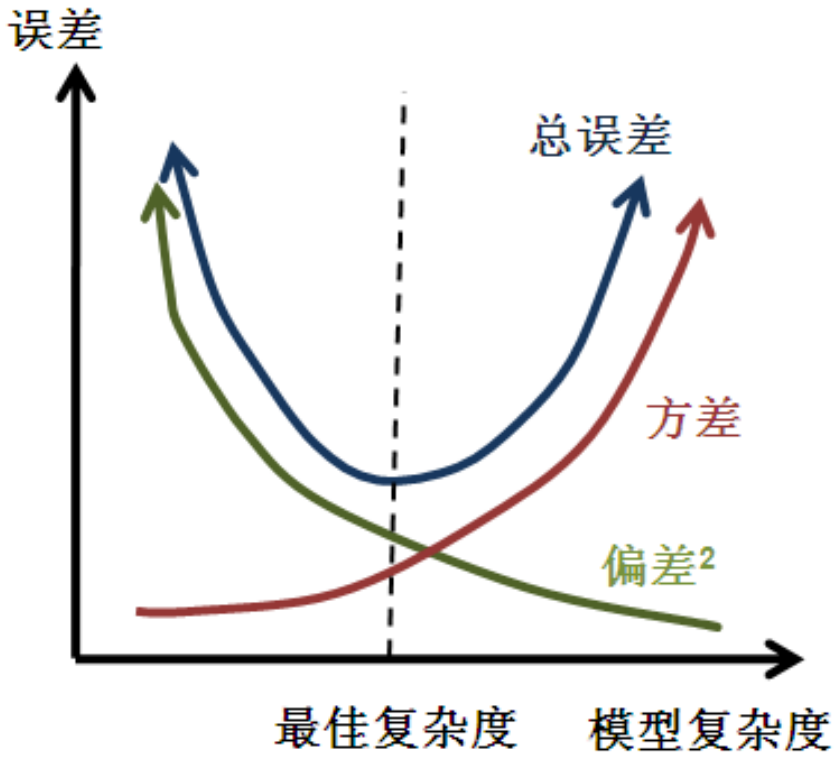
$$Bias(\hat{y}) = E(\hat{y} - y)$$

$$Var(\hat{y}) = E(\hat{y}^2) - E(\hat{y})^2$$

  - 简单说,偏差和方差都对误差有贡献.
  - 增加模型复杂度时,偏差减少,方差增加

误差/偏差/方差

- 模型的总误差形成抛物线形状



例:过拟合

```
from sklearn.neighbors import
  KNeighborsClassifier
from sklearn.datasets import load_iris
iris = load_iris()
X,y = iris.data, iris.target

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X,y)
knn.score(X,y)
1.0
```

- 训练和测试是一个数据集
- 所以要将数据集划分为训练集和测试集

案例:偏差/方差权衡

- 附文档

数据科学基础
(With Python)

改善模型

交叉验证(1)

- k折交叉验证
  - 数据随机等分为k个不相交子集
  - 总共执行k次训练与测试
    - 第i折: 使用D_i作为测试集, 其他作为训练集
  - k =3/5/10等
- 每个样本有k-1次作为训练样本,仅有一次作为测试样本
- 正确率=k次迭代中正确分类的总次数/数据集样本数

交叉验证(2)

- 留一法: k折,其中k =样本总数.
  - 适用小规模数据
- 分层交叉验证: 每折是分层的, 使得各折的类分布与初始数据近似相同

交叉验证(3)

- 交叉验证效果上相当于在同一个数据集上进行了多次训练集测试集划分.
- 交叉验证的好处是能最准确评估模型的来自样本的误差
  - 比简单的训练集-测试集划分好

交叉验证与模型性能

- 当交叉验证误差和训练误差都很高时,发生欠拟合
- 当交叉验证误差高而训练误差低时,发生过拟合
- 当交叉验证误差低,且只比训练误差略高时,拟合正好.

Bootstrap样本

- 对训练样本进行一致放回抽样
  - 每次抽出的样本放回训练集,从而有同等可能性被再次抽中
  - 对小数据集很有效
- 常用的 .632 boostrap
  - 设数据集有d个样本, 对数据集做d次可放回抽样, 产生大小为d的训练集.
  - 没有选入训练集的数据形成测试集
  - 平均来说约63.2%的初始数据被选中, 剩下的36.8%形成测试集

例:Bootstrap样本

```
pop = np.arange(1,21)
print pop
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

print np.random.choice(pop,size=20,replace=True)
[ 6, 12, 13,  9, 10, 12,  6, 16,  1, 17,  2, 13,  8, 14,  7, 19,  6, 19, 12, 11]
```

网格搜索(1)

- 网格搜索:通过蛮力试验不同参数组合,帮助我们选择最佳模型.

```
from sklearn.grid_search import GridSearchCV
knn = KNeighborsClassifier()  # 创建未指定k的kNN模型
ks = range(1,30,2) # k的取值范围
param_grid = dict(n_neighbors=ks)
# 即{"n_neighbors: [1,3,5...29]}
grid = GridSearchCV(knn,param_grid,cv=5,
                    scoring='accuracy')
grid.fit(X,y)
# 对15种k值各执行5折交叉验证
# 建立了15*5=75个kNN模型
```

网格搜索(2)

- 结果

```
grid.grid_scores_    # 15次的评分数组

[result[1] for result in grid.grid_scores_]
# 15个平均得分

grid.best_params_
grid.best_scores_
grid.best_estimator_
```

网格搜索(3)

- 更多参数组合

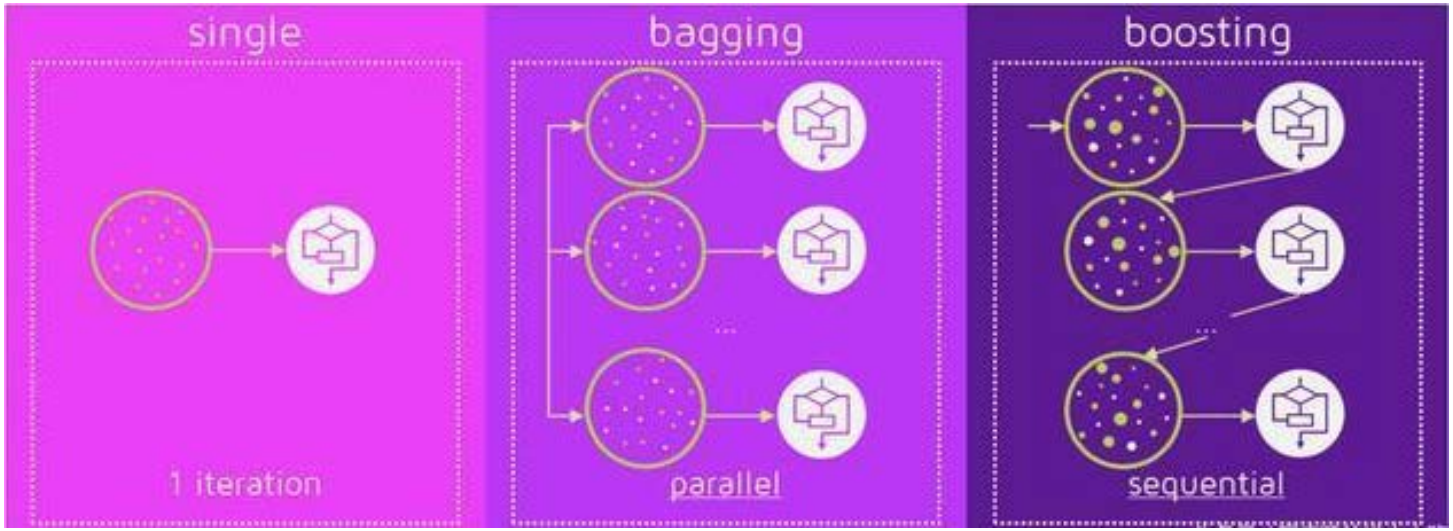
```
knn = KNeighborsClassifier()
ks = range(1,30)
algo = ['kd_tree','ball_tree','auto','brute']
ps = range(1,8)
ws = ['uniform','distance']
param_grid = dict(n_neighbors=ks,
                  weights=ws,
                  algorithm=algo,
                  p=ps)
grid = GridSearchCV(knn,param_grid,cv=5,
                    scoring='accuracy')
grid.fit(X,y)
```

交叉验证案例

- 附文档

集成方法

- 从一系列模型 $M_1, M_2, \ldots, M_k$, 创建组合模型M*
  - 可提高正确率
- 方法:Bagging, Boosting



例:集成(1)

- 考虑二类分类问题(预测0/1)

```
m1 = np.random.rand(1000)
m2 = np.random.rand(1000)
m3 = np.random.rand(1000)
m4 = np.random.rand(1000)
m5 = np.random.rand(1000)
# 模拟5个模型:大于3则预测为1类
pred1 = np.where(m1>0.3,1,0)
pred2 = np.where(m2>0.3,1,0)
pred3 = np.where(m3>0.3,1,0)
pred4 = np.where(m4>0.3,1,0)
pred5 = np.where(m5>0.3,1,0)
```

例:集成(2)

- 各模型单独预测有70%左右正确率

```
print pred1.mean()
0.699
print pred2.mean()
0.698
print pred3.mean()
0.71
print pred4.mean()
0.699
print pred5.mean()
0.685
```

例:集成(3)

- 组合模型正确率更高

```
# 小技巧
ensemble_pred = np.round(
(pred1+pred2+pred3+pred4+pred5)/5.0).astype(int)

# 5个预测值中有3个以上是1,平均值舍入后才为1
# 模拟多数投票

ensemble_pred.mean()
0.83
```

- 如果增加模型,误差将减少.

Bagging思想

- 比方:基于多个医生的投票多数作出诊断
- **Bagging = Bootstrap aggregation**
  - **Bootstrap即前面说的抽样方式**

Bagging过程

- 给定数据集D,大小为N
  - 在第i 次迭代, 从D可放回抽样N个样本形成训练集D_i (即boostrap样本)
  - 从D_i 训练模型M_i
  - 各M_i对未知样本X分类:返回预测的类
  - Bagged模型M*计算投票数, 将票数最多的类赋予X

Bagging特点

- Bagging通过降低模型方差改善性能
- 准确度
  - 通常显著优于单个分类器
  - 对噪声数据: 不会明显恶化, 更鲁棒
- 预测: 可应用于连续值的预测, 对给定样本取各预测值的平均值
  - 在预测中具有更好的准确度

Bagging演示

- 附文档

Boosting思想

- 比方:咨询多个医生, 对其诊断加权,组合.
  - 下一个医生注意上一个医生的诊断错误
  - 医生权重基于医生过去的诊断正确率

Boosting过程(1)

- 训练样本初始权重$w_i=1/N$
- 训练模型M_1
  - 计算错误率(加权) $r_1 = \frac{\sum w_i \times abs(y_i - \hat{y}_i)}{\sum w_i}$
  - 根据r_1计算M_1的权重α_1
  - 根据α_1调整样本权重 $w_i = w_i \times \exp(\alpha_1 \times abs(y_i - \hat{y}_i))$
    - 被M_1分类错误的样本权重增大
- 训练下一个模型M_2
  - 样本权重使得M_2更重视被M_1误分类的样本

Boosting过程(2)

- 重复上述过程,构造一系列模型

  $M_1, M_2, ..., M_k$
- 最终的M* 组合了各M_i的投票
  - 各模型投票的权重取决于其正确率

Boosting特点

- 模型之间存在依赖关系,所以需要按顺序学习模型
- 能将多个弱分类器转化成一个强分类器
- 与**bagging**的比较: **boosting**一般能得到更高的准确度, 但是也有模型过拟合的风险

Boosting演示

- 附文档

End